33장 7번째 데이터 타입 Symbol

33.1 심벌이란?

심벌은 변경 불가능한 원시 타입의 값입니다. 주로 **이름의 충돌 위험이 없는 유일한 프로퍼티 키를 만들기 위해** 사용한다.

프로퍼티 키로 사용할 수 있는 값은 빈 문자열을 포함하는 모든 문자열 또는 심벌 값

33.2 심벌 값의 생성

33.2.1 Symbol 함수

심벌 값은 Symbol 함수를 호출하여 생성해야 한다.

```
const mySymbol = Symbol(); // 절대 중복되지 않는 유일무이한 값을
생성
console.log(typeof mySymbol); // symbol

// 심벌 값은 외부로 노출되지 않아 확인할 수 없다.
console.log(mySymbol); // Symbol()

new Symbol(); // TypeError new 연산자와 함께 생성하지 않는다.
```

Symbol 함수에는 선택적으로 문자열을 인수로 전달할 수 있는데, 이는 디버깅 용도로만 사용되며 심벌 값 생성에 어떠한 영향도 주지 않는다.

```
// 심벌 값에 대한 설명이 같더라도 중복되지 않는 심벌 값을 생성
const mySymbol1 = Symbol('mySymbol');
const mySymbol2 = Symbol('mySymbol');
console.log(mySymbol1 === mySymbol2); // false
```

심벌 값도 객체처럼 접근하면 암묵적으로 래퍼 객체를 생성한다.

```
const mySymbol = Symbol('mySymbol');

// 심벌도 래퍼 객체를 생성한다.
// description 프로퍼티와 toString 메서드는 Symbol.prototype의
```

```
프로퍼티다.
console.log(mySymbol.description); // mySymbol
console.log(mySymbol.toString()); // Symbol(mySymbol)
```

심벌 값은 암묵적으로 문자열이나 숫자 타입으로 변환되지 않지만 불리언 타입으로는 변환된다.

```
const mySymbol = Symbol();

console.log(mySymbol + ''); // TypeError 문자열 변환 X
console.log(+mySymbol); // TypeError 숫자 타입 변환 X
console.log(!!mySymbol); // true 불리언 타입으로 암묵적 변환
// if문 등에서 존재 확인이 가능하다.
if(mySymbol) console.log('mySymbol is not empty');
```

32.2.2 Symbol.for / Symbol.keyFor 메서드

Symbol.for 메서드는 인수로 전달받은 문자열을 키로 사용하여 해당 키와 일치하는 심벌 값을 검색한다.

```
// 전역 심벌 레지스트리에 mySymbol이라는 키로 저장된 심벌 값이 없으면
📌새로 생성
const s1 = Symbol.for('mySymbol');
// 전역 심벌 레지스트리에 mySymbol이라는 키로 저장된 심벌 값이 있으면
📌해당 값을 반환
const s2 = Symbol.for('mySymbol');

console.log(s1 === s2); // true
```

Symbol.keyFor 메서드를 사용하면 전역 심벌 레지스트리에 저장된 심벌 값의 키를 추출할 수 있다.

```
// 전역 심벌 레지스트리에 mySymbol 키로 저장된 심벌 값이 없으면 새로
생성
const s1 = Symbol.for('mySymbol');
// 심벌 값의 키를 추출
Symbol.keyFor(s1); // mySymbol

// 전역 심벌 레지스트리에 등록되어 관리되지 않음
```

```
const s2 = Symbol('foo');
Symbol.keyFor(s2); // undefined
```

33.3 심벌과 상수

값에는 특별한 의미가 없고 상수 이름 자체에 의미가 있는 경우 변경/중복될 가능성이 없는 심벌 값을 사용할 수 있다.

```
// 위, 아래, 왼쪽, 오른쪽을 나타내는 상수를 정의한다.
// 중복될 가능성이 없는 심벌 값으로 상수 값을 생성한다.
const Direction = {
  UP: Symbol('up'),
  DOWN: Symbol('down'),
  LEFT: Symbol('left'),
  RIGHT: Symbol('right')
};

const myDirection = Direction.UP;

if(myDirection === Direction.UP) {
  console.log('you are going up');
}
```

33.4 심벌과 프로퍼티 키

객체의 프로퍼티 키는 동적으로 생성할 수도 있다.

```
// 심벌 값을 프로퍼티 키로 사용하려면 대괄호를 사용해야 한다.
// 다른 프로퍼티 키와 절대 충돌하지 않는다.
const obj = {
  [Symbol.for('mySymbol')]: 1
};

// 프로퍼티에 접근할 때도 마찬가지로 대괄호를 사용해야 한다.
obj[Symbol.for('mySymbol')]; // 1
```

33.5 심벌과 프로퍼티 은닉

심벌 값을 프로퍼티 키로 사용하여 생성한 프로퍼티는 for ...in문이나 Object.keys, Object.getOwnPropertyNames 메서드로 찾을 수 없다.

외부에 노출할 필요가 없는 프로퍼티를 은닉할 수 있다.

하지만 프로퍼티를 완전하게 숨길 수 있는 것은 아니며, 새롭게 도입된 Object.getOwnPropertySymbols 메서드를 사용하여 프로퍼티를 찾을 수 있다.

```
const obj = {
  // 심벌 값으로 프로퍼티 키를 생성
  [Symbol('mySymbol')]: 1
};

for(const key in obj) {
  console.log(key); // 아무것도 출력되지 않음
}

console.log(Object.keys(obj)); // []
console.log(Object.getOwnPropertyNames(obj)); // []

//getOwnPropertySymbols 메서드는 인수로 전달한 객체의 심벌 프로퍼티 키를 배열로 반환
console.log(Object.getOwnPropertySymbols(obj)); // [Symbol(mySymbol)]

//getOwnPropertySymbols 메서드로 심벌 값도 찾을 수 있음
const symbolKey1 = Object.getOwnPropertySymbols(obj)[0];
console.log(obj[symbolKey1]); // 1
```

33.6 심벌과 표준 빌트인 객체 확장

일반적으로 표준 빌트인 객체에 사용자 정의 메서드를 직접 추가하여 확장하는 것은 권장하지 않으며 읽기 전용으로 사용하는 것이 좋다.

> 미래에 표준 사양으로 추가될 메서드의 이름이 중복될 수 있기 때문

하지만 중복될 가능성이 없는 심벌 값으로 프로퍼티 키를 생성하여 표준 빌트인 객체를 확장하면 어떤 프로퍼티 키와도 충돌할 위험이 없어 안전하게 확장 가능

```
Array.prototype[Symbol.for('sum')] = function () {
  return this.reduce((acc, cur) => acc + cur, 0);
```

```
};

[1, 2][Symbol.for('sum')](); // 3
```

33.7 Well-known Symbol

자바스크립트가 기본 제공하는 빌트인 심벌 값

빌트인 심벌 값은 Symbol 함수의 프로퍼티에 할당되어 있다.